



5th GI/ITG KuVS Fachgespräch on NG SDPs

Experiments and analysis on Hypervisor-Based Fault Tolerance in Virtualized Environments

*P. Baglietto, M. Maresca, **M. Stecca***

*CIPI - Computer Platform Research Center
University of Genoa and Padua, Italy*

Munich, 11th October 2011



Outline

- Introduction
- The Fault Tolerance in the VMware Virtualization system
- Experiments
 - Basic experiments
 - I/O-intensive applications
 - CPU-intensive applications
- Case Study
 - Service Composition Engine in Virtualized Environments relying on the VMware's FT module
- Conclusion



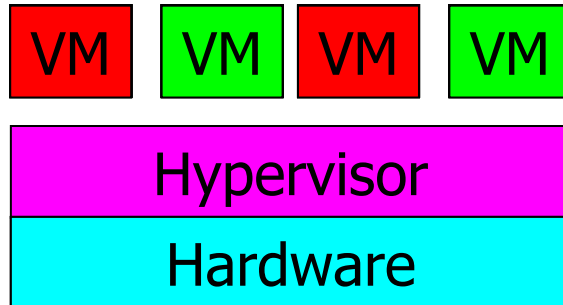
Introduction (1/3)

- The Virtualization technology is widely used in data centers as well as it enables the Cloud Computing paradigm
- Fault Tolerance (FT) is an important requirement in many software systems (e.g., business applications, Telco apps, etc.)
- Two approaches for managing FT in virtualized systems
 - Checkpointing (e.g., XEN)
 - Record & Replay (e.g., VMware)

Introduction (2/3)

Which kind of failures are we talking about?

- Software failures?
- Hardware failures?
- Knee failures?



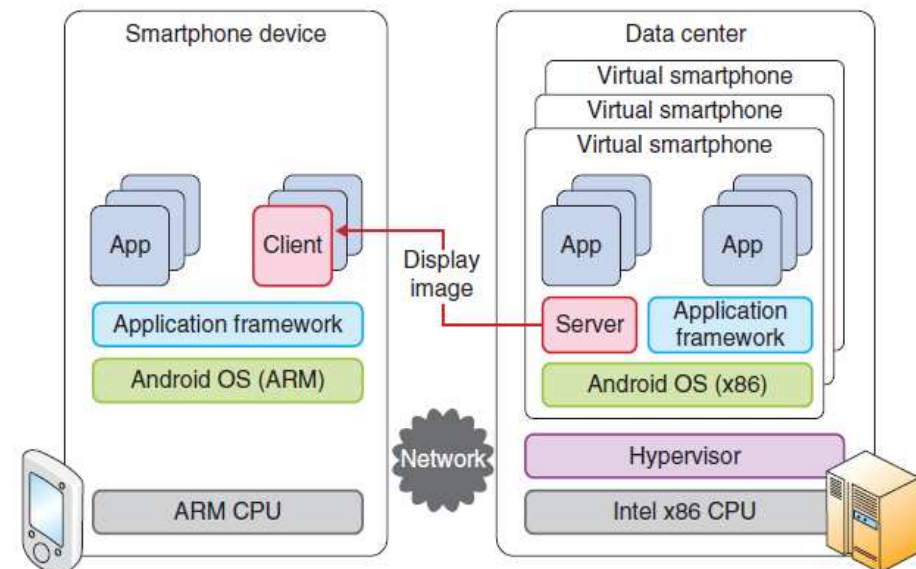
The FT property refers to VMs
Managed at Hypervisor level
Against hardware failures

VM = Not Protected **VM** = Protected

Introduction (3/3)

The Virtualization technology is usually adopted in Server-side solutions... Is there any relationship with mobile applications?

Common idea behind the "CloudPhone" (Intel) and the "Virtual Smartphone over IP" projects (NTT) → mobile devices should be supported by Server-Side platforms to compensate their limited on-board resources (e.g., CPU, memory, storage, battery energy, etc.)

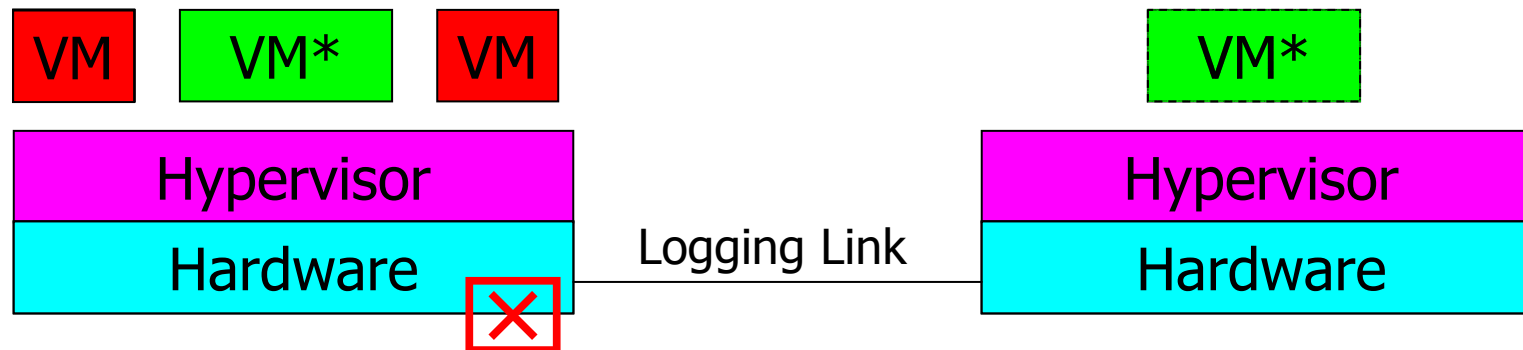


Source: Mitsutaka Itoh et al., "Virtual Smartphone over IP", NTT Technical Review. Vol. 8 No. 7 July 2010.

The FT in the VMware system

The Hypervisor based Fault Tolerance mechanism provided by VMware

- The two hosts are connected through Logging Traffic network link.
- Two VMs run in lockstep on two distinct physical hosts.
- The two VMs run independently but only the Primary VM actually communicates with the external world (BUT nondeterministic events go from the Primary VM to the Secondary VM).
- In case of a failure of the first physical host, the Secondary VM replaces the Primary VM and continues the execution of the applications installed in the first VM in a transparent way and with zero downtime*.



* This is the difference respect of "VMware High Availability"

Experiments

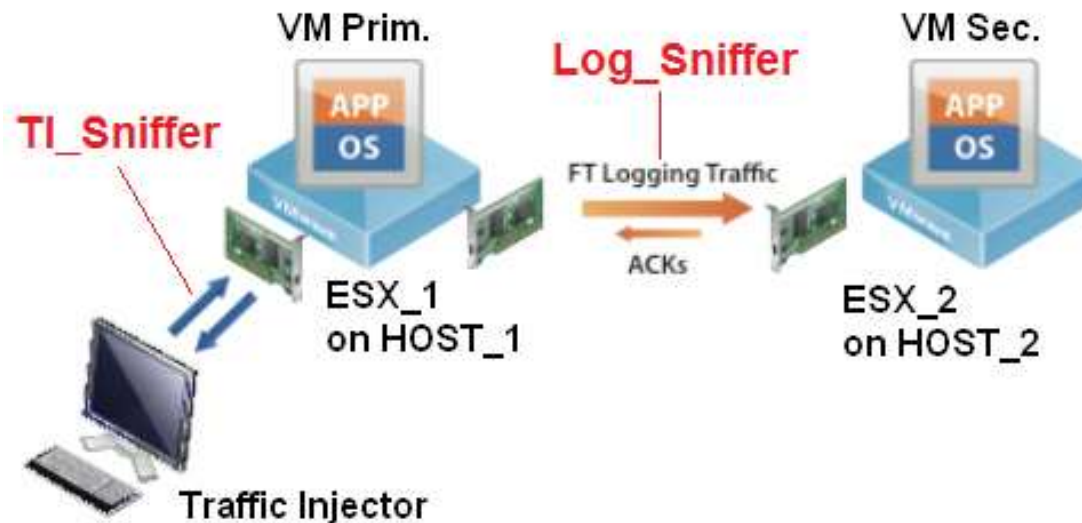
Hardware Setup

Physical Hosts: IBM Blade Center HS22 (2 Intel E5530 processors at 2,4 GHz, 32GB RAM)

Hypervisor: ESX (VMware vSphere)

Guest OS: Red Hat Enterprise Linux 5.4 64 bit

Sniffers: both the Traffic Injector Sniffer TI_Sniffer and the Logging Sniffer Log_Sniffer run Wireshark (promiscuous mode)





Experiment #1 (1/3)

Title: Management of a single TCP connection

Goal: Understanding how the FT module deals with network events

Software Setup:

- 1 TCP Server running on the Primary VM
- 1 Traffic Injector running outside the FT setup
- 2 Sniffers (i.e., TI_Sniffer and Log_Sniffer)

Methodology:

- TCP communications from the TI to the TCP Server
- We cross-check the packets captured by the two Sniffers

Experiment #1 (2/3)

Original TCP SYN message captured by TI_Sniffer

```
0000 00 50 56 bd 38 e9 00 0c 29 69 ac bf 08 00 45 00 .PV.8... )i....E.
0010 00 30 0b e5 40 00 80 06 04 b2 ac 11 c9 07 ac 11 .0..@... .....
0020 c9 06 23 8c 13 88 cb d4 47 7a 00 00 00 00 70 02 ..#. .... Gz....p.
0030 fa f0 53 9a 00 00 02 04 05 b4 01 01 04 02 ..S. ....
```

"Forwarded" TCP SYN message captured by Log_Sniffer

```
0060 ff ff 0f c1 10 00 a4 9b 9c 75 2c 00 00 00 04 00 ..... u, .....
0070 06 f5 f7 22 67 36 00 00 00 00 1d b3 06 80 ff ff ... "g6.. .....
0080 ff ff 00 00 00 00 00 00 00 00 09 00 00 00 00 00 ..... +. "
0090 00 00 00 00 00 00 8e 00 00 00 15 00 01 2b f7 22 .....
00a0 67 36 00 00 00 00 1d b3 06 80 ff ff ff ff 00 00 g6.....
00b0 00 00 00 00 00 00 01 01 01 00 00 00 00 00 00 00 .....
00c0 00 00 3e 00 00 00 00 00 1e 00 ff ff ff ff 04 1b ..>.....
00d0 4c a8 11 00 00 00 00 00 04 00 00 00 00 00 01 ff L.....
00e0 ff ff ff ff ff ff 00 50 56 bd 38 e9 00 0c 29 69 ..... P v.8... )i
00f0 ac bf 08 00 45 00 00 30 0b e5 40 00 80 06 04 b2 ..... E..0 ..@....
0100 ac 11 c9 07 ac 11 c9 06 23 8c 13 88 cb d4 47 7a ..... #. .... Gz
0110 00 00 00 00 70 02 fa f0 53 9a 00 00 02 04 05 b4 ..... p... S. ....
0120 01 01 04 02 2c 00 00 00 0e 00 01 32 2c 23 67 36 ..... ,. ...2, #g6
0130 00 00 00 00 ff ff ff ff ff ff ff ff ff ff ff ff .....
0140 ff ff ff ff 0f c1 10 00 22 77 a9 75 5c 90 aa 75 ..... "w.u\...u
0150 8a 00 00 00 15 00 01 00 2b 2b 67 36 00 00 00 00 ..... ++g6....
0160 45 42 20 88 ff ff ff ff 1f 00 00 00 00 00 00 00 EB ..... \k*2
0170 05 01 00 00 01 00 00 00 0e 00 01 00 5c 6b 2a 32 .....
0180 01 00 1f 00 ff ff ff ff 51 1b 4c a8 11 00 00 00 ..... Q.L.....
0190 01 00 00 00 1f 00 00 00 01 00 00 00 02 00 a8 c8 .....
01a0 42 00 00 00 00 00 00 3e 00 00 00 00 00 00 00 B..... >.....
```

Experiment #1 (3/3)

The flow of the messages related to a single TCP communication

	Traffic Injector	Primary VM	Secondary VM
t ₀	SYN →		
t ₁		SYN* →	
t ₂			← ACK**
t ₃		← SYN+ACK	
t ₄	ACK →		
t ₅		ACK* →	
t ₆			← ACK**
t ₇	PUSH →		
t ₈		PUSH* →	
t ₉			← ACK**
t ₁₀		← ACK	

where

ACK = “Real” ACK

ACK* = Forwarded ACK
(VM1 → VM2)

ACK** = “Log” ACK
(VM2 → VM1)



Experiment #2 (1/4)

Title: Simulation of a Primary VM HW failure during an intensive TCP traffic generation

Goal: Understanding how the FT module manages a hardware failure during intensive network operations

Software Setup: see Experiment #1

Methodology:

- TCP communications from the TI to the TCP Server
- We turn off HOST_1 abruptly
- We check the logs in all the involved machines searching for exceptions, network errors, etc.
- We cross-check two counters: *sentMessages* in TI and *receivedMessages* in TCP Server
- We check the packets captured by the TI_Sniffer



Experiment #2 (2/4)

Results

- We obtained ***sentMessages = receivedMessages***
- Number of Java exceptions in the Traffic Injector: **0**
- Number of Java exceptions in the TCP Server: **0**
- Number of errors in the Traffic Injector: **0**
- Number of errors in the TCP Server: **0**

**The transition from HOST_1 to
HOST_2 happened transparently**





Experiment #2 (3/4)

BUT... What happened behind the scenes?

112258	35.156085	172.17.201.7	172.17.201.6	TCP	30432	> 5000	[SYN]	Seq=1296902418	Len=0	MSS=1460
112259	35.156294	172.17.201.7	172.17.201.6	TCP	30433	> 5000	[SYN]	Seq=4047721983	Len=0	MSS=1460
112260	35.158065	172.17.201.7	172.17.201.6	TCP	30434	> 5000	[SYN]	Seq=3895520901	Len=0	MSS=1460
112261	35.159089	172.17.201.7	172.17.201.6	TCP	30435	> 5000	[SYN]	Seq=3220825321	Len=0	MSS=1460
112262	35.159105	172.17.201.7	172.17.201.6	TCP	30436	> 5000	[SYN]	Seq=2255836328	Len=0	MSS=1460
112263	35.159371	172.17.201.7	172.17.201.6	TCP	30437	> 5000	[SYN]	Seq=3250683081	Len=0	MSS=1460
112264	35.168114	172.17.201.7	172.17.201.6	TCP	30438	> 5000	[SYN]	Seq=3432547114	Len=0	MSS=1460
112265	35.175412	172.17.201.7	172.17.201.6	TCP	30440	> 5000	[SYN]	Seq=3172543472	Len=0	MSS=1460
112266	35.175412	172.17.201.7	172.17.201.6	TCP	30439	> 5000	[SYN]	Seq=3342455417	Len=0	MSS=1460
112267	35.176665	172.17.201.7	172.17.201.6	TCP	30441	> 5000	[SYN]	Seq=3351700990	Len=0	MSS=1460
112268	35.179717	172.17.201.7	172.17.201.6	TCP	30442	> 5000	[SYN]	Seq=443022226	Len=0	MSS=1460
112269	35.179743	172.17.201.7	172.17.201.6	TCP	30443	> 5000	[SYN]	Seq=2979920441	Len=0	MSS=1460
112270	35.180560	172.17.201.7	172.17.201.6	TCP	30444	> 5000	[SYN]	Seq=1247450914	Len=0	MSS=1460
112271	35.182809	172.17.201.7	172.17.201.6	TCP	30445	> 5000	[SYN]	Seq=2220111893	Len=0	MSS=1460

1. Logs recorded by the TISniffer during the transition phase: a block of SYN messages is waiting for SYN+ACK replies from the TCP Server

Experiment #2 (4/4)

112298	35.007903	172.17.201.7	172.17.201.0	TCP	29091 > 5000 [FIN, ACK] Seq=104040034 ACK=193500310 Win=0 Len=0
112299	35.686433	fe80::2885:cd29:b00f:ff02::1:2		DHCPv6	solicit
112300	35.842625	Cisco_52:97:90	Spanning-tree-(for-br	STP	Conf. Root = 8192/00:d0:04:36:cc:ab Cost = 65 Port = 0x8010
112301	36.188340	vmware_bd:38:e9	Broadcast	RARP	who is 00:50:56:bd:38:e9? Tell 00:50:56:bd:38:e9
112302	36.188346	vmware_bd:38:e9	Broadcast	RARP	who is 00:50:56:bd:38:e9? Tell 00:50:56:bd:38:e9
112303	36.188347	vmware_bd:38:e9	Broadcast	RARP	who is 00:50:56:bd:38:e9? Tell 00:50:56:bd:38:e9
112304	36.227723	fe80::21a:64ff:fe85:2	ff02::1:2	DHCPv6	solicit
112305	36.638120	172.17.200.25	172.17.223.255	BROWSER	Browser Election Request
112306	36.995214	vmware_bd:38:e9	Broadcast	RARP	who is 00:50:56:bd:38:e9? Tell 00:50:56:bd:38:e9
112307	37.176883	00:21:5e:46:53:19	Broadcast	ARP	who has 172.17.192.30? Tell 172.17.193.134
112308	37.599224	172.17.193.18	172.17.223.255	BROWSER	Host Announcement XPSP2, Workstation, Server, NT Workstation, Potential B
112309	37.848422	Cisco_52:97:90	Spanning-tree-(for-br	STP	Conf. Root = 8192/00:d0:04:36:cc:ab Cost = 65 Port = 0x8010
112310	37.979021	172.17.201.7	172.17.201.6	TCP	30434 > 5000 [SYN] Seq=3895520901 Len=0 MSS=1460

2. When the Secondary VM takes over there is a network reconfiguration phase.
3. When the TCP ACK timeout expires, the TI sends the SYN messages again. The Secondary machine will reply to those messages.



Experiment #3 (1/2)

Title: Analysis of an I/O intensive application

Goal: Understanding the impact of the FT module usage on I/O-bound applications

Software Setup: see Experiment #1

Methodology:

- TCP communications from the TI to the TCP Server
- We chose two evaluation parameters:
 - **Avg_Serving_Time:** the average time taken by the Server to serve one request;
 - **Throughput:** the number of requests served per second.
- We measured the two parameters two times:
 - FT On
 - FT Off



Experiment #3 (2/2)

Results

	Avg_Serving_Time (ms)	Throughput (reqs/sec)
FT on	0.0221	232435
FT off	0.0196	233202

→ The performance decrease is acceptable



Experiment #4 (1/3)

Title: Analysis of a CPU intensive application

Goal: Understanding the impact of the FT module usage on CPU-bound applications

Software Setup: see Experiment #1 without the TI

Methodology:

- We ran a CPU bound program in Primary VM
- We chose one evaluation parameter:
 - ***Exec_Time***: the average execution time for the CPU-bound program;
- We measured Exec_Time two times:
 - FT On
 - FT Off



Experiment #4 (2/3)

The program

Input values: N (number of for loop executions) and BigNumber (number of iterations in the inner loop)

```
1 While (n_executions++<N)  {
2  startTime=gettimeofday;
3  for i=0 to BigNumber{
4      uselessVariable = gettimeofday;
5      set of mathematical operations;}//end for
6  endTime=gettimeofday;
7  times[n_executions] = endTime-startTime;}//end while
8  result = mean (times, n_executions);
```



Experiment #4 (3/3)

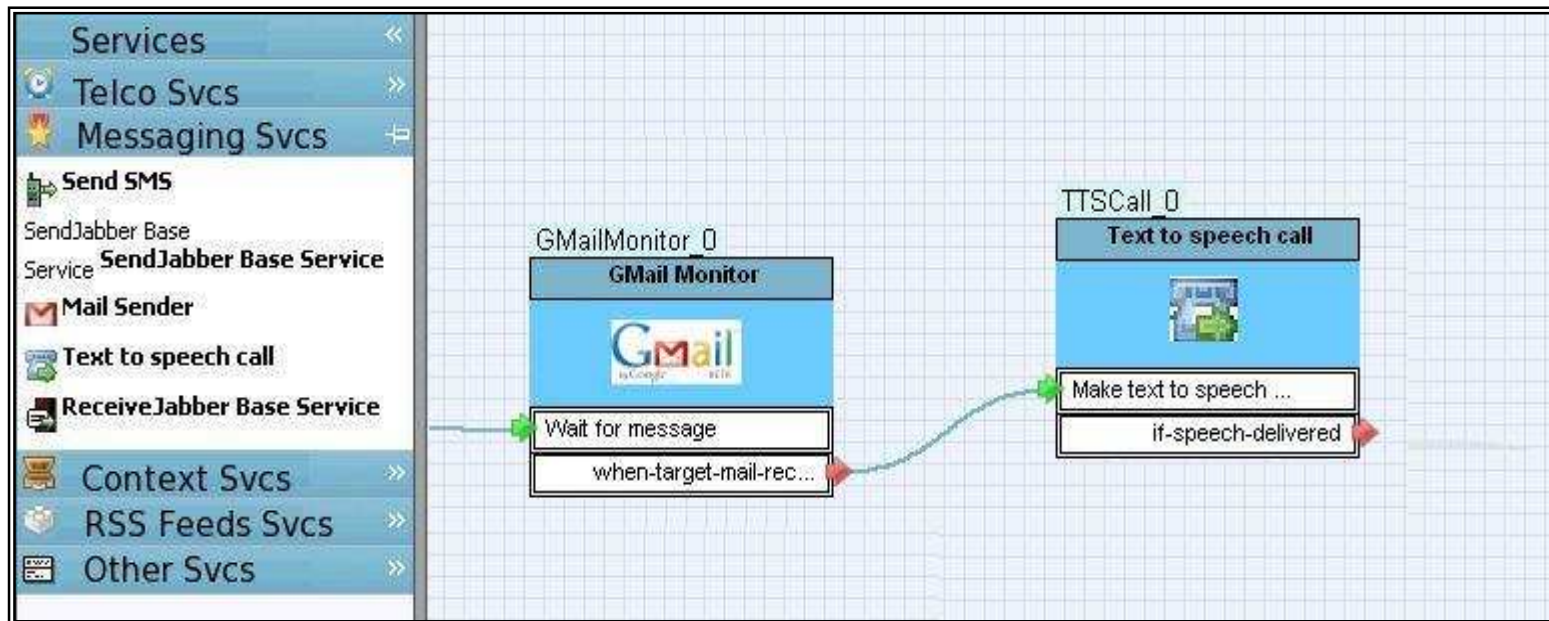
Results

	Result (ms)	Log_BW (MB/s)
FT Enabled	2107	1
FT not Enabled	227	Not Appl.

→ If we reduce/remove the `gettimeofday` method invocations, the performances improve

Case Study (1/3)

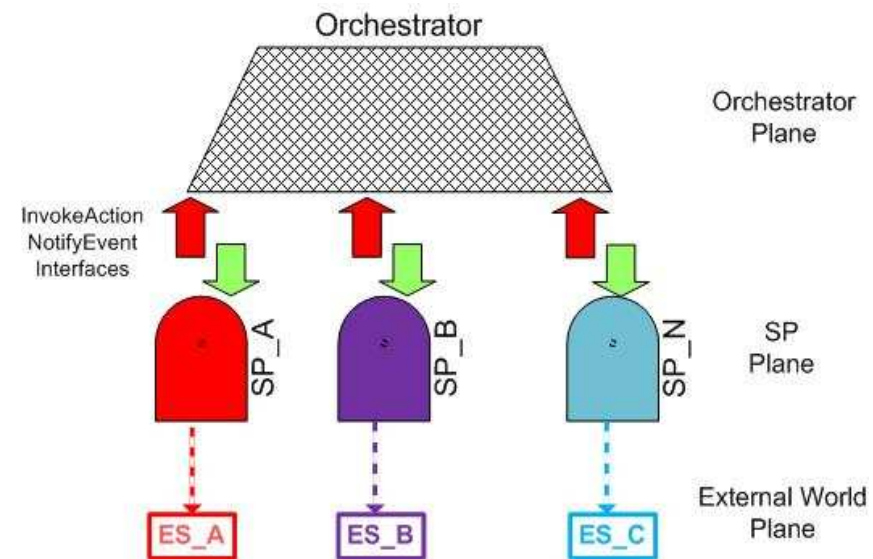
- We tested a Service Composition engine
- Composite Services (a.k.a., Mashups) can be created by means of this graphical tool



Case Study (2/3)

Architecture:

- *External Services – ESs:*
Web APIs available over the Internet
- *Service Proxies – SPs:*
wrap an ES (adaptation layer)
- *Orchestrator:* executes the logic of Composite Services



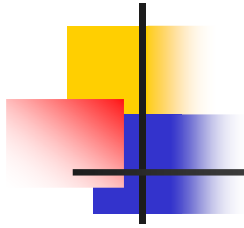


Case Study (3/3)

Evaluation tests:

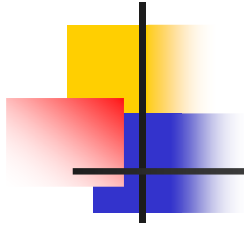
- *Latency*: the time needed to execute one CS
- *Throughput*: the rate of events managed by the engine
- *Note*: we performed the stress-test 2 times (FT on vs FT off)
- *Results*: the performance degradation is negligible

	<i>Latency Avg (μs)</i>	<i>Throughput (Events/sec)</i>	<i>Log. Traffic (KB/sec)</i>
<i>FT On</i>	82	15948870	761
<i>FT Off</i>	60	16187340	Not Applicable



Conclusions

- We tested the hypervisor-based FT mechanism provided by VMware (record & replay)
- We analyzed in depth how it works
- We performed some evaluation tests
 - I/O-bound apps: OK
 - CPU-bound apps: OK without gettimeofday calls
- We analyzed the Service Composition engine case study
- Future works:
 - Testing real world applications
 - Comparing it with checkpointing-based systems



Q&A

Thank you!

Questions?

Follow me on Twitter: @steccami